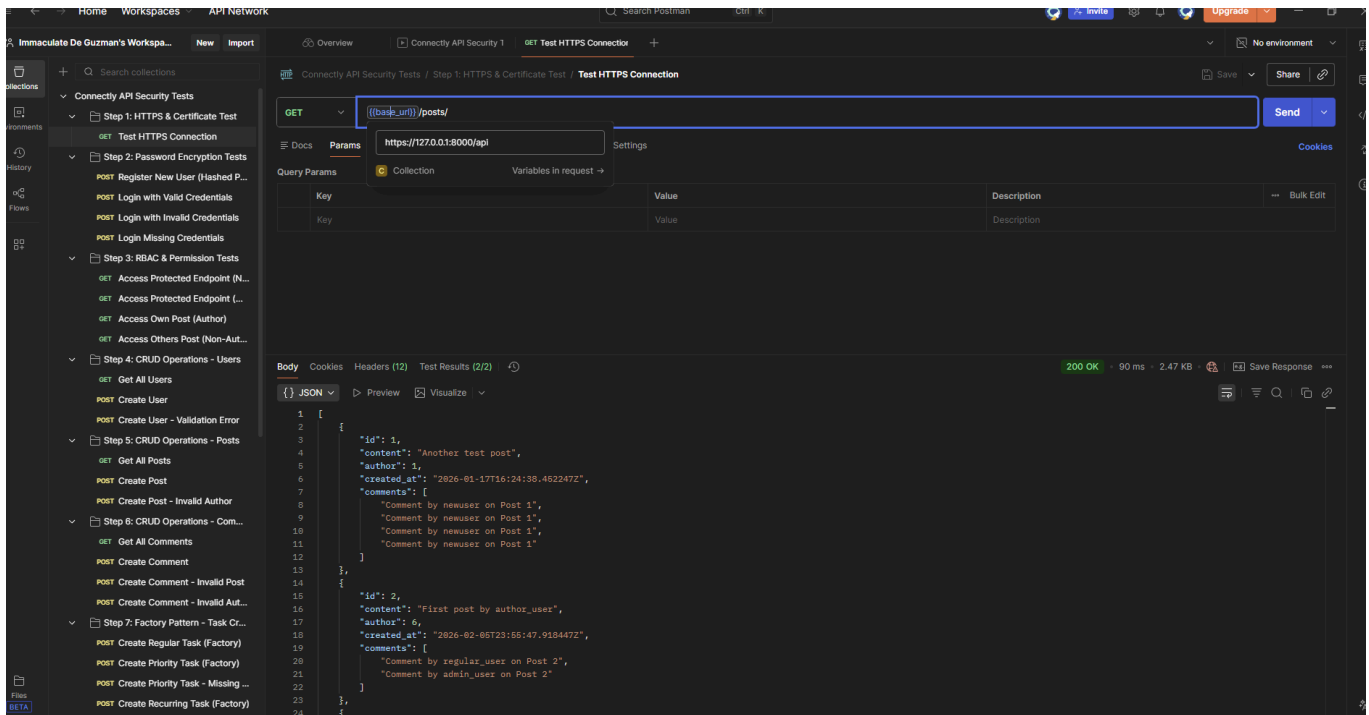


Step 1: HTTPS & Certificate Test

Test HTTPS Connection <https://127.0.0.1:8000/api/posts/>

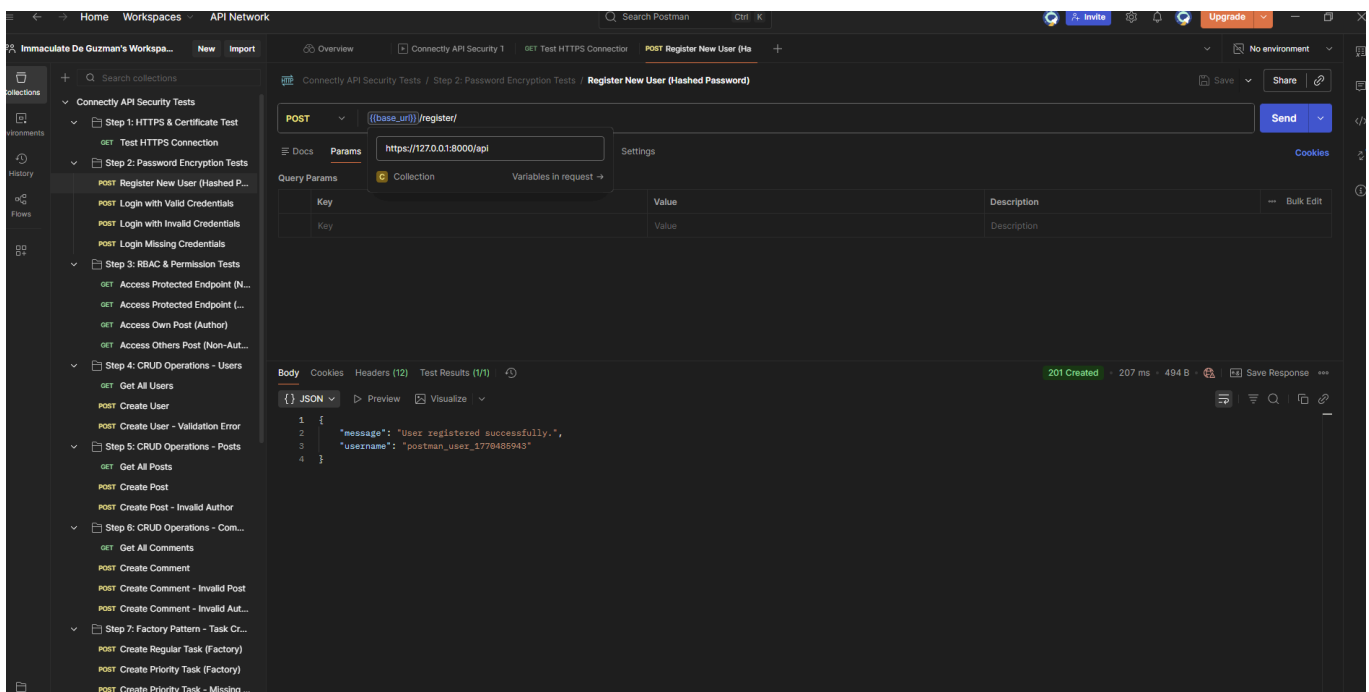


The screenshot shows the Postman interface with a workspace named "Immaculate De Guzman's Workspa...". The left sidebar displays a collection of API tests under "Connectivity API Security Tests". The main panel shows a "GET" request to "https://127.0.0.1:8000/api/posts/" with a "Send" button. The response is a 200 OK status with a response time of 90 ms and a body of 2.47 KB. The response body is a JSON array of two posts:

```
1 {
2   {
3     "id": 1,
4     "content": "Another test post",
5     "author": "1",
6     "created_at": "2026-01-17T16:24:30.452247Z",
7     "comments": [
8       "Comment by newuser on Post 1",
9       "Comment by newuser on Post 1",
10      "Comment by newuser on Post 1",
11      "Comment by newuser on Post 1"
12    ]
13   },
14   {
15     "id": 2,
16     "content": "First post by author_user",
17     "author": "6",
18     "created_at": "2026-02-06T23:55:47.918447Z",
19     "comments": [
20       "Comment by regular_user on Post 2",
21       "Comment by admin_user on Post 2"
22     ]
23   }
24 }
```

Step 2: Password Encryption Tests

Register New User (Hashed Password) <https://127.0.0.1:8000/api/register/>



The screenshot shows the Postman interface with the same workspace. The left sidebar shows the "POST Register New User (Hashed Password)" test selected. The main panel shows a "POST" request to "https://127.0.0.1:8000/api/register/" with a "Send" button. The response is a 201 Created status with a response time of 207 ms and a body of 494 B. The response body is a JSON object:

```
1 {
2   "message": "User registered successfully.",
3   "username": "postman_user_1778485943"
4 }
```

Login with Valid Credentials https://127.0.0.1:8000/api/login/

The screenshot shows the Postman interface with a REST client request configured for a POST method to the endpoint `https://127.0.0.1:8000/api/login/`. The request body is in JSON format, containing the following data:

```
1 {
2   "message": "Authentication successful!",
3   "username": "admin_user"
4 }
```

The response status is **200 OK**, with a response time of 214 ms and a body size of 473 B. The response body is also in JSON format:

```
1 {
2   "message": "Authentication successful!",
3   "username": "admin_user"
4 }
```

Login with Invalid Credentials https://127.0.0.1:8000/api/login/

The screenshot shows the Postman interface with a REST client request configured for a POST method to the endpoint `https://127.0.0.1:8000/api/login/`. The request body is in JSON format, containing the following data:

```
1 {
2   "error": "Invalid credentials."
3 }
```

The response status is **401 Unauthorized**, with a response time of 214 ms and a body size of 451 B. The response body is in JSON format:

```
1 {
2   "error": "Invalid credentials."
3 }
```

Login Missing Credentials https://127.0.0.1:8000/api/login/

The screenshot shows the Postman interface with a REST client request configured for a POST method to the endpoint `https://127.0.0.1:8000/api/login/`. The request body is in JSON format, containing the following data:

```
1 {
2   "error": "Username and password are required."
3 }
```

The response status is **400 Bad Request**, with a response time of 24 ms and a body size of 465 B. The response body is in JSON format:

```
1 {
2   "error": "Username and password are required."
3 }
```

Step 3: RBAC & Permission Tests

Access Protected Endpoint (No Token) <https://127.0.0.1:8000/api/protected/>

The screenshot shows the Postman interface with the 'Access Protected Endpoint (No Token)' test selected. The request is a GET to `{{base_url}}/protected/` with the URL parameter set to `https://127.0.0.1:8000/api`. The response is a 401 Unauthorized status with a response time of 28 ms and 499 B. The JSON body contains the message: `"detail": "Authentication credentials were not provided."`

```
1 {
2   "detail": "Authentication credentials were not provided."
3 }
```

Access Protected Endpoint (With Token) <https://127.0.0.1:8000/api/protected/>

The screenshot shows the Postman interface with the 'Access Protected Endpoint (With Token)' test selected. The request is a GET to `{{base_url}}/protected/` with the URL parameter set to `https://127.0.0.1:8000/api`. The response is a 200 OK status with a response time of 14 ms and 434 B. The JSON body contains the message: `"message": "Authenticated!"`

```
1 {
2   "message": "Authenticated!"
3 }
```

Access Own Post (Author) <https://127.0.0.1:8000/api/posts/4/>

The screenshot shows the Postman interface with the 'Access Own Post (Author)' test selected. The request is a GET to `{{base_url}}/posts/4/` with the URL parameter set to `https://127.0.0.1:8000/api`. The response is a 200 OK status with a response time of 16 ms and 438 B. The JSON body contains the content: `"content": "Post by admin_user"`

```
1 {
2   "content": "Post by admin_user"
3 }
```

Access Others Post (Non-Author) <https://127.0.0.1:8000/api/posts/4/>

The screenshot shows the Postman interface with a GET request to `https://127.0.0.1:8000/api/posts/4/`. The response is a 403 Forbidden status with a message: `"detail": "You do not have permission to perform this action."`

Key	Value	Description
Key	Value	Description

```
{
  "detail": "You do not have permission to perform this action."
}
```

Step 4: CRUD Operations - Users Get All Users <https://127.0.0.1:8000/api/users/>

The screenshot shows the Postman interface with a GET request to `https://127.0.0.1:8000/api/users/`. The response is a 200 OK status with a JSON array of three user objects.

Key	Value	Description
Key	Value	Description

```
[
  {
    "id": 1,
    "username": "newuser",
    "email": "newuser@example.com",
    "created_at": "2026-01-17T16:04:58.872461Z"
  },
  {
    "id": 2,
    "username": "Jed",
    "email": "jengiaja@example.com",
    "created_at": "2026-01-17T16:12:14.941586Z"
  },
  {
    "id": 3,
    "username": "Ralph",
    "email": "ralphnocom@example.com",
    "created_at": "2026-01-22T13:48:01.167486Z"
  }
]
```

Create User <https://127.0.0.1:8000/api/users/>

The screenshot shows the Postman interface with a POST request to `https://127.0.0.1:8000/api/users/`. The response is a 201 Created status with a JSON object representing the newly created user.

Key	Value	Description
Key	Value	Description

```
{
  "id": 10,
  "username": "crud_test_user_1778486617",
  "email": "crudtest1778486617@example.com",
  "created_at": "2026-02-07T17:58:16.778248Z"
}
```

Create User - Validation Error <https://127.0.0.1:8000/api/users/>

The screenshot shows the Postman interface with a collection named "Immaculate De Guzman's Workspa...". The selected request is "POST Create User - Validation Error" under the "Step 4: CRUD Operations - Users" folder. The URL is `https://127.0.0.1:8000/api/users/`. The response is a 400 Bad Request with a status of 25 ms and 510 B. The JSON body of the response is:

```
1 {
2   "username": [
3     "This field is required."
4   ],
5   "email": [
6     "Enter a valid email address."
7   ]
8 }
```

Step 5: CRUD Operations - Posts Get All Posts <https://127.0.0.1:8000/api/posts/>

The screenshot shows the Postman interface with the same collection. The selected request is "GET All Posts" under the "Step 5: CRUD Operations - Posts" folder. The URL is `https://127.0.0.1:8000/api/posts/`. The response is a 200 OK with a status of 61 ms and 2.47 KB. The JSON body of the response is:

```
1 [
2   {
3     "id": 1,
4     "content": "Another test post",
5     "author": 1,
6     "created_at": "2026-01-17T16:24:38.452247Z",
7     "comments": [
8       "Comment by newuser on Post 1",
9       "Comment by newuser on Post 1",
10      "Comment by newuser on Post 1",
11      "Comment by newuser on Post 1"
12    ]
13  },
14  {
15    "id": 2,
16    "content": "First post by author_user",
17    "author": 6,
18    "created_at": "2026-02-08T23:55:47.918447Z",
19    "comments": [
20      "Comment by regular_user on Post 2",
21      "Comment by admin_user on Post 2"
22    ]
23  }
24 ]
```

Create Post https://127.0.0.1:8000/api/posts/

The screenshot shows the Postman interface with a workspace named "Immaculate De Guzman's Workspa...". The active collection is "Step 5: CRUD Operations - Posts", and the selected request is "POST Create Post". The URL is set to "https://127.0.0.1:8000/api/posts/". The request body is in JSON format, containing the following data:

```
1 {
2   "id": 16,
3   "content": "This is a test post created via Postman at 1770486799",
4   "author": 1,
5   "created_at": "2026-02-07T17:03:19.268939Z",
6   "comments": []
7 }
```

The response status is "201 Created" with a response time of 67 ms and a body size of 569 B. The response body is in JSON format, containing the following data:

```
1 {
2   "id": 16,
3   "content": "This is a test post created via Postman at 1770486799",
4   "author": 1,
5   "created_at": "2026-02-07T17:03:19.268939Z",
6   "comments": []
7 }
```

Create Post - Invalid Author https://127.0.0.1:8000/api/posts/

The screenshot shows the Postman interface with a workspace named "Immaculate De Guzman's Workspa...". The active collection is "Step 5: CRUD Operations - Posts", and the selected request is "POST Create Post - Invalid Author". The URL is set to "https://127.0.0.1:8000/api/posts/". The request body is in JSON format, containing the following data:

```
1 {
2   "author": [
3     "Invalid pk \"999999\" - object does not exist."
4   ]
5 }
```

The response status is "400 Bad Request" with a response time of 25 ms and a body size of 489 B. The response body is in JSON format, containing the following data:

```
1 {
2   "author": [
3     "Invalid pk \"999999\" - object does not exist."
4   ]
5 }
```

Step 6: CRUD Operations - Comments

Get All Comments <https://127.0.0.1:8000/api/comments/>

The screenshot shows the Postman interface with a GET request to `https://127.0.0.1:8000/api/comments/`. The response is a 200 OK status with a JSON body containing an array of three comments. The left sidebar shows the collection structure, and the top bar indicates the environment is 'No environment'.

Key	Value	Description
Key	Value	Description

```
1 [
2   {
3     "id": 1,
4     "text": "This is a comment.",
5     "author": 1,
6     "post": 1,
7     "created_at": "2026-01-24T17:43:14.437943Z"
8   },
9   {
10    "id": 2,
11    "text": "Great post!",
12    "author": 6,
13    "post": 2,
14    "created_at": "2026-02-08T23:05:47.965288Z"
15  },
16  {
17    "id": 3,
18    "text": "Thanks for sharing",
19    "author": 4,
20    "post": 2,
21    "created_at": "2026-02-08T23:05:47.978146Z"
22  }
23 ]
```

Create Comment <https://127.0.0.1:8000/api/comments/>

The screenshot shows the Postman interface with a POST request to `https://127.0.0.1:8000/api/comments/`. The response is a 201 Created status with a JSON body containing a single comment object. The left sidebar shows the collection structure, and the top bar indicates the environment is 'No environment'.

Key	Value	Description
Key	Value	Description

```
1 {
2   "id": 7,
3   "text": "This is a test comment created via Postman",
4   "author": 1,
5   "post": 1,
6   "created_at": "2026-02-07T17:57:32.102586Z"
7 }
```

Create Comment - Invalid Post <https://127.0.0.1:8000/api/comments/>

The screenshot shows the Postman interface with a POST request to `https://127.0.0.1:8000/api/comments/`. The response is a 400 Bad Request status with a JSON body containing an error message. The left sidebar shows the collection structure, and the top bar indicates the environment is 'No environment'.

Key	Value	Description
Key	Value	Description

```
1 {
2   "post": [
3     "Invalid pk \"99999\" - object does not exist."
4   ]
5 }
```

Create Comment - Invalid Author <https://127.0.0.1:8000/api/comments/>

The screenshot shows a Postman interface for a workspace named "Immaculate De Guzman's Workspa...". The active collection is "Step 6: CRUD Operations - Comments", and the selected request is "POST Create Comment - Invalid Author". The URL is set to `{{base_url}}/comments/`. The request body is a JSON object:

```
{ 1: { 2:   "author": { 3:     "invalid_pk \"99999\" - object does not exist." 4:   } 5: }
```

The response status is "400 Bad Request" with a response time of 32 ms and a body size of 489 B. The response body is empty.

Step 7: Factory Pattern - Task Creation Create Regular Task (Factory) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows a Postman interface for the same workspace. The active collection is "Step 7: Factory Pattern - Task Creation", and the selected request is "POST Create Regular Task (Factory)". The URL is set to `https://127.0.0.1:8000/api/tasks/create/`. The request body is a JSON object:

```
{ 1: { 2:   "message": "Task created successfully!", 3:   "task_id": 13, 4:   "task_type": "regular", 5:   "default_priority": "Medium" 6: }
```

The response status is "201 Created" with a response time of 85 ms and a body size of 518 B. The response body is empty.

Create Priority Task (Factory) <https://127.0.0.1:8000/api/tasks/create/>

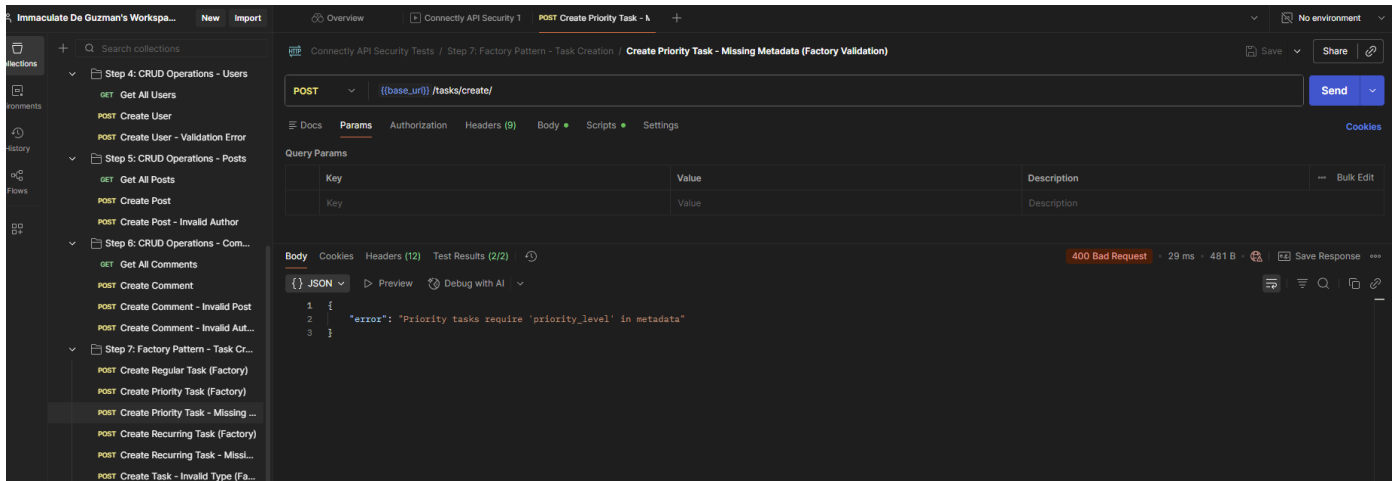
The screenshot shows a Postman interface for the same workspace. The active collection is "Step 7: Factory Pattern - Task Creation", and the selected request is "POST Create Priority Task (Factory)". The URL is set to `https://127.0.0.1:8000/api/tasks/create/`. The request body is a JSON object:

```
{ 1: { 2:   "message": "Task created successfully!", 3:   "task_id": 14, 4:   "task_type": "priority", 5:   "default_priority": "Medium" 6: }
```

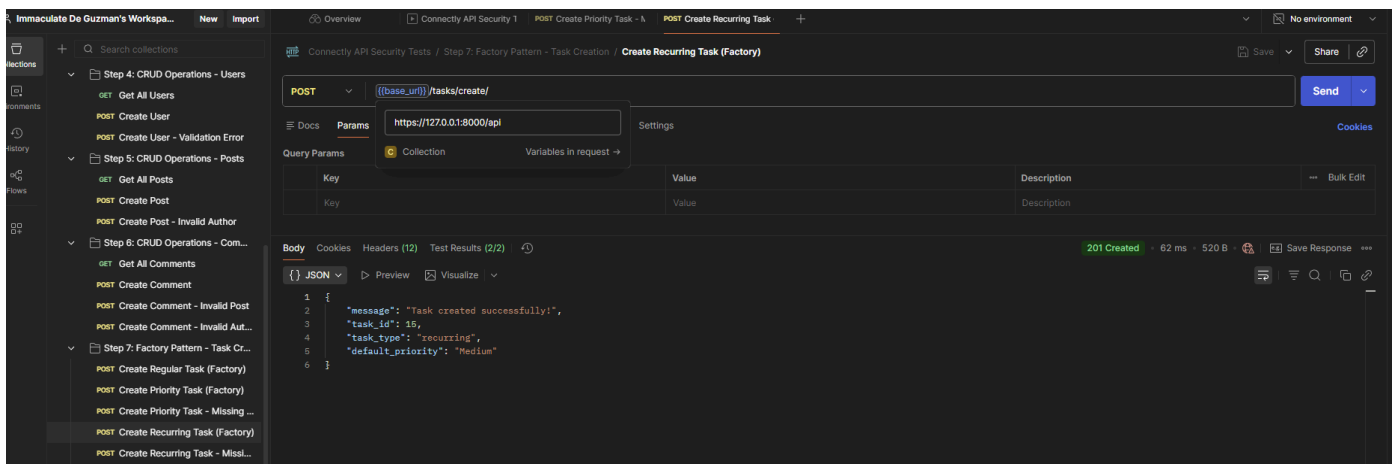
The response status is "201 Created" with a response time of 58 ms and a body size of 519 B. The response body is empty.

Create Priority Task - Missing Metadata (Factory Validation)

<https://127.0.0.1:8000/api/tasks/create/>

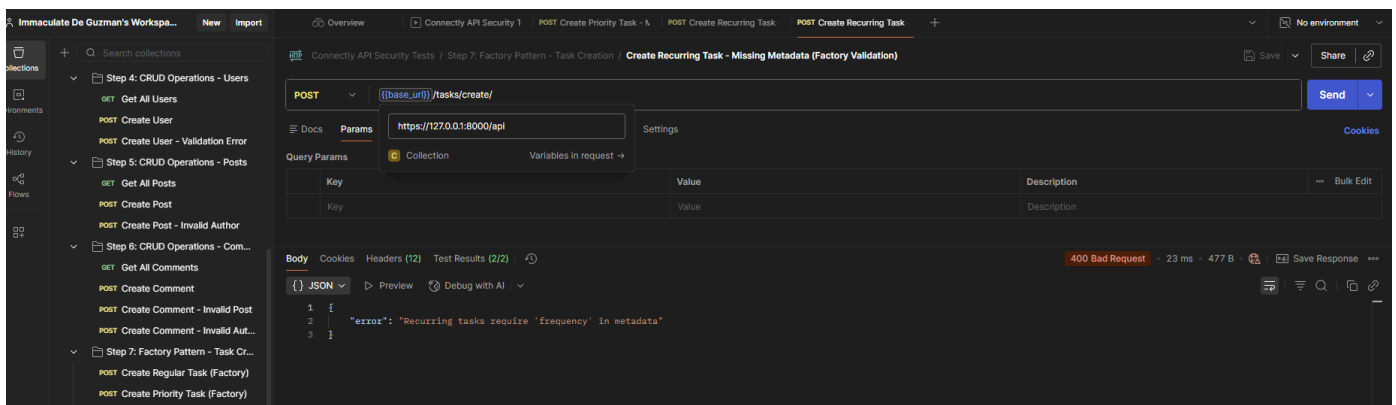


Create Recurring Task (Factory) <https://127.0.0.1:8000/api/tasks/create/>



Create Recurring Task - Missing Metadata (Factory Validation)

<https://127.0.0.1:8000/api/tasks/create/>



Create Task - Invalid Type (Factory Validation) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows a Postman collection named "Immaculate De Guzman's Workspa..." with a folder "Step 4: CRUD Operations - Users". The selected request is "POST Create Task - Invalid Type (Factory Validation)". The URL is "https://127.0.0.1:8000/api/tasks/create/". The response is a 400 Bad Request with the following JSON body:

```
{
  "error": "Invalid task type. Must be one of: ['regular', 'priority', 'recurring']"
}
```

Create Task - Invalid User (Relational Validation) <https://127.0.0.1:8000/api/tasks/create/>

The screenshot shows a Postman collection named "Immaculate De Guzman's Workspa..." with a folder "Step 4: CRUD Operations - Users". The selected request is "POST Create Task - Invalid User (Relational Validation)". The URL is "https://127.0.0.1:8000/api/tasks/create/". The response is a 400 Bad Request with the following JSON body:

```
{
  "error": "Assigned user not found."
}
```

Step 8: CRUD Operations - Tasks Get All Tasks <https://127.0.0.1:8000/api/tasks/>

The screenshot shows a Postman collection named "Immaculate De Guzman's Workspa..." with a folder "Step 8: CRUD Operations - Tasks". The selected request is "GET Get All Tasks". The URL is "https://127.0.0.1:8000/api/tasks/". The response is a 200 OK with the following JSON body:

```
[
  {
    "id": 15,
    "title": "Weekly Status Report",
    "description": "A recurring task that runs weekly",
    "assigned_to": 1,
    "task_type": "recurring",
    "metadata": {
      "frequency": "weekly"
    },
    "completed": false,
    "created_at": "2026-02-07T18:04:17.292573Z",
    "updated_at": "2026-02-07T18:04:17.292612Z"
  },
  {
    "id": 14,
    "title": "High Priority Task",
    "description": "An urgent task requiring priority_level metadata",
    "assigned_to": 1,
    "task_type": "priority",
    "metadata": {
      "priority_level": "High"
    },
    "completed": false,
    "created_at": "2026-02-07T18:02:18.929685Z",
    "updated_at": "2026-02-07T18:02:18.929726Z"
  },
  {
    "id": 13,
    "title": "Regular Text Task",
    "description": "A regular task created via Postman to test Factory Pattern",
    "assigned_to": 1,
    "task_type": "regular",
    "metadata": {}
  }
]
```

Get Tasks - Filter by Type (Priority) <https://127.0.0.1:8000/api/tasks/?type=priority>

The screenshot shows the Postman interface with a GET request to `https://127.0.0.1:8000/api/tasks/?type=priority`. The request is successful (200 OK) and returns a JSON array of three tasks. The first two tasks are "High Priority Task" and the third is "Weekly Status Report".

Key	Value	Description
type	priority	

```
1 [
2   {
3     "id": 14,
4     "title": "High Priority Task",
5     "description": "An urgent task requiring priority_level metadata",
6     "assigned_to": 1,
7     "task_type": "priority",
8     "metadata": {
9       "priority_level": "high"
10    },
11    "completed": false,
12    "created_at": "2026-02-07T18:02:18.029852Z",
13    "updated_at": "2026-02-07T18:02:18.029726Z"
14  },
15  {
16    "id": 18,
17    "title": "High Priority Task",
18    "description": "An urgent task requiring priority_level metadata",
19    "assigned_to": 1,
20    "task_type": "priority",
21    "metadata": {
22      "priority_level": "high"
23    },
24    "completed": false,
25    "created_at": "2026-02-07T17:36:55.489476Z",
26    "updated_at": "2026-02-07T17:36:55.489997Z"
27  },
28  {
29    "id": 6,
30    "title": "Weekly Status Report",
31    "description": "A recurring task that runs weekly",
32    "assigned_to": 1,
33    "task_type": "recurring",
34    "metadata": {
35      "frequency": "weekly"
36    },
37    "completed": false,
38    "created_at": "2026-02-07T17:36:56.623641Z",
39    "updated_at": "2026-02-07T17:36:56.623578Z"
40  }
41 ]
```

Get Tasks - Filter by Type (Recurring) <https://127.0.0.1:8000/api/tasks/?type=recurring>

The screenshot shows the Postman interface with a GET request to `https://127.0.0.1:8000/api/tasks/?type=recurring`. The request is successful (200 OK) and returns a JSON array of three tasks. The first two tasks are "Weekly Status Report" and the third is "High Priority Task".

Key	Value	Description
type	recurring	

```
1 [
2   {
3     "id": 15,
4     "title": "Weekly Status Report",
5     "description": "A recurring task that runs weekly",
6     "assigned_to": 1,
7     "task_type": "recurring",
8     "metadata": {
9       "frequency": "weekly"
10    },
11    "completed": false,
12    "created_at": "2026-02-07T18:04:17.292576Z",
13    "updated_at": "2026-02-07T18:04:17.292612Z"
14  },
15  {
16    "id": 11,
17    "title": "Weekly Status Report",
18    "description": "A recurring task that runs weekly",
19    "assigned_to": 1,
20    "task_type": "recurring",
21    "metadata": {
22      "frequency": "weekly"
23    },
24    "completed": false,
25    "created_at": "2026-02-07T17:36:56.623641Z",
26    "updated_at": "2026-02-07T17:36:56.623578Z"
27  },
28  {
29    "id": 7,
30    "title": "Weekly Status Report",
31    "description": "A recurring task that runs weekly",
32    "assigned_to": 1,
33    "task_type": "recurring",
34    "metadata": {
35      "frequency": "weekly"
36    },
37    "completed": false,
38    "created_at": "2026-02-07T17:36:56.623641Z",
39    "updated_at": "2026-02-07T17:36:56.623578Z"
40  }
41 ]
```

Get Task Detail <https://127.0.0.1:8000/api/tasks/13>

Postman interface showing the GET Task Detail endpoint. The URL is `https://127.0.0.1:8000/api/tasks/13`. The response is a 200 OK status with a JSON body containing task details.

```
1 {
2   "id": 13,
3   "title": "Regular Test Task",
4   "description": "A regular task created via Postman to test Factory Pattern",
5   "assigned_to": 1,
6   "task_type": "regular",
7   "metadata": {},
8   "completed": false,
9   "created_at": "2026-02-07T10:01:03.649050Z",
10  "updated_at": "2026-02-07T10:01:03.649981Z"
11 }
```

Update Task <https://127.0.0.1:8000/api/tasks/13>

Postman interface showing the PUT Update Task endpoint. The URL is `https://127.0.0.1:8000/api/tasks/13`. The response is a 200 OK status with a JSON body containing updated task details.

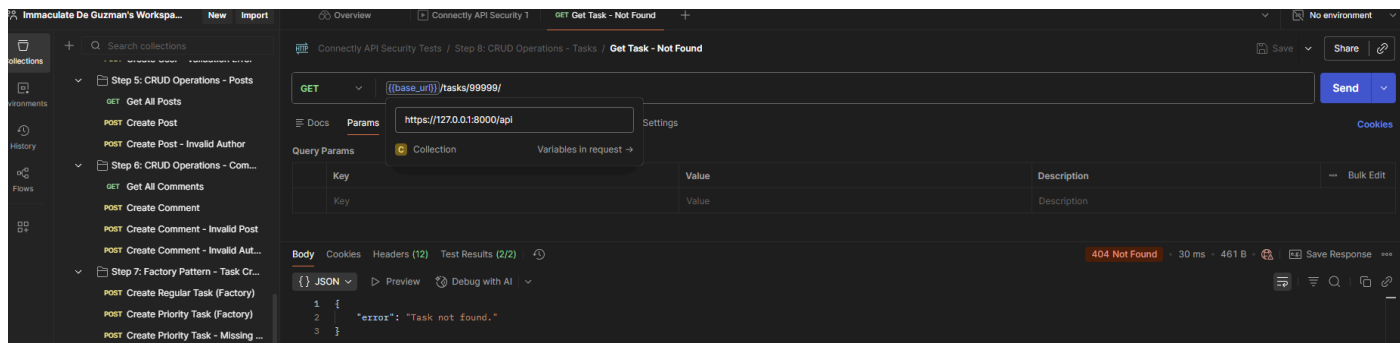
```
1 {
2   "id": 13,
3   "title": "Updated Task Title",
4   "description": "A regular task created via Postman to test Factory Pattern",
5   "assigned_to": 1,
6   "task_type": "regular",
7   "metadata": {},
8   "completed": true,
9   "created_at": "2026-02-07T10:01:03.649050Z",
10  "updated_at": "2026-02-07T10:13:59.189688Z"
11 }
```

Delete Task <https://127.0.0.1:8000/api/tasks/13>

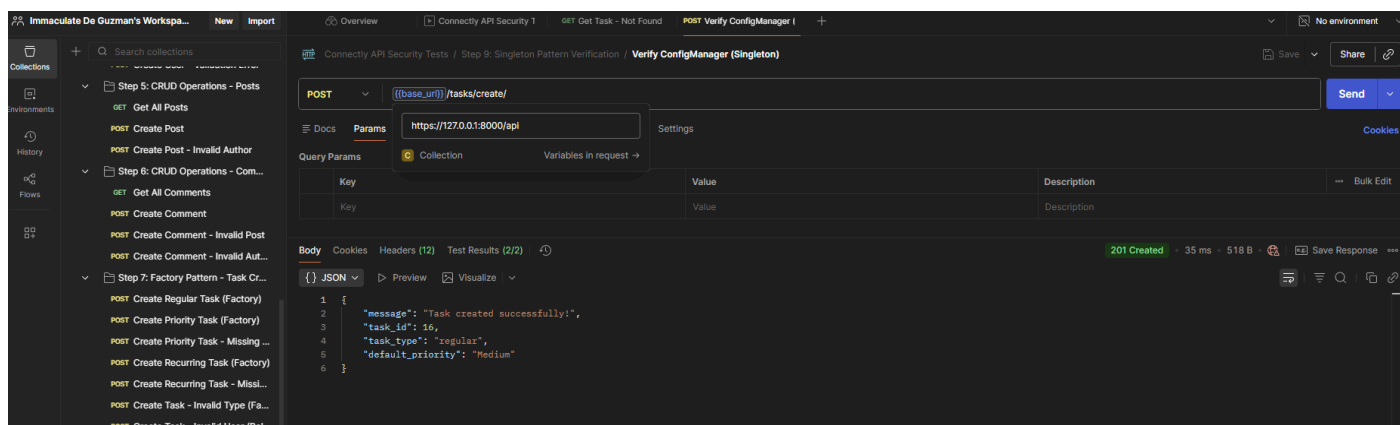
Postman interface showing the DELETE Task endpoint. The URL is `https://127.0.0.1:8000/api/tasks/13`. The response is a 204 No Content status.

```
1 {
2   "message": "Task deleted successfully."
3 }
```

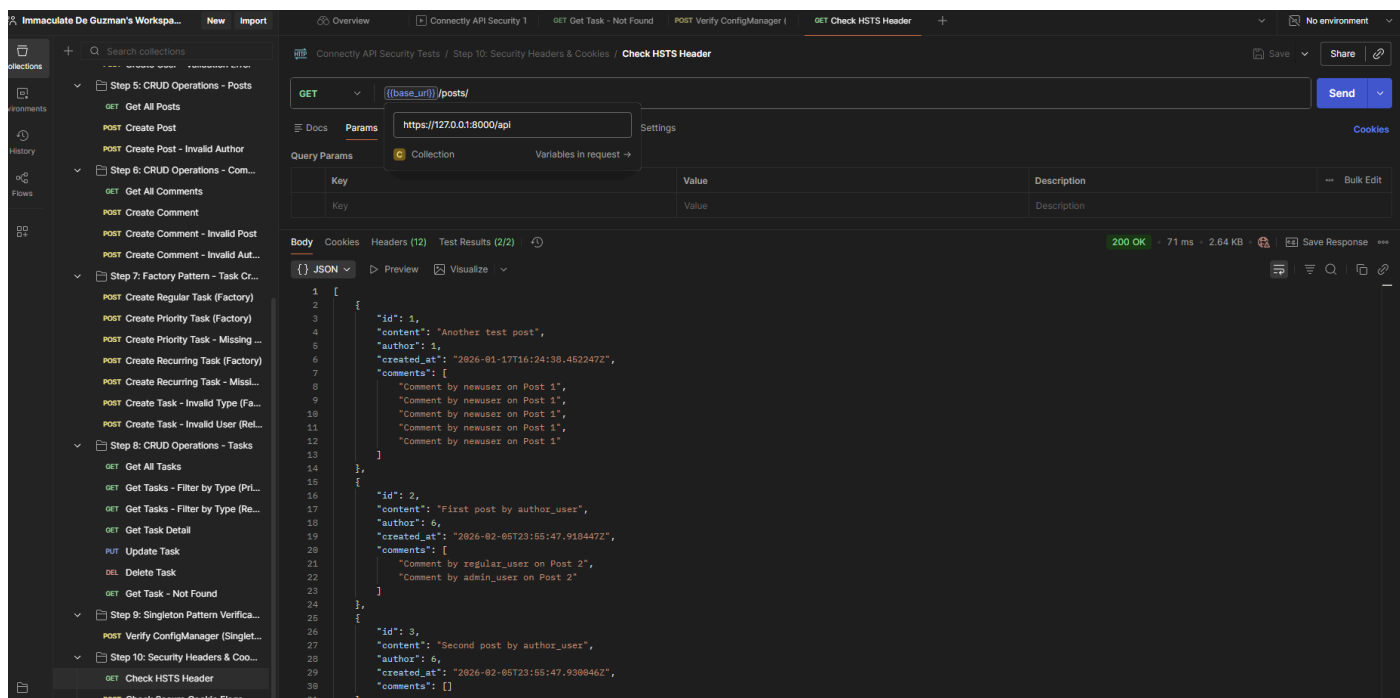
Get Task - Not Found <https://127.0.0.1:8000/api/tasks/99999/>



Step 9: Singleton Pattern Verification Verify ConfigManager (Singleton) <https://127.0.0.1:8000/api/tasks/create/>



Step 10: Security Headers & Cookies Check HSTS Header <https://127.0.0.1:8000/api/posts/>



Check Secure Cookie Flags <https://127.0.0.1:8000/api/login/>

The screenshot shows the Postman interface with a POST request to `https://127.0.0.1:8000/api/login/`. The request is configured with the following details:

- Method:** POST
- URL:** `https://127.0.0.1:8000/api/login/`
- Params:** A single parameter with key `url` and value `https://127.0.0.1:8000/api`.
- Query Params:** A table with two columns: Key and Value. The first row has Key `url` and Value `https://127.0.0.1:8000/api`.
- Body:** A JSON object:

```
{  "message": "Authentication successful!",  "username": "admin_user"}
```

The response is a 200 OK status with a response time of 190 ms and a body size of 473 B. The response body is displayed in JSON format:

```
{  "message": "Authentication successful!",  "username": "admin_user"}
```